CAJ:mww 11/11/08 1023089 303978.01

Attorney Reference Number 3382-64706-01
Application Number 10/607,591

INFORMAL COMMUNICATION – DO NOT ENTER

PLEASE DELIVER DIRECTLY TO EXAMINER THUY CHAN DAO
Fax No.: (571) 273-8570
Total No. Pages: 7 including this cover sheet
Message: Transmitted herewith is an Interview Agenda. If you do not receive all pages, or if you have problems receiving transmittal, please call Mark W. Wilson at (503) 595-5300.

In re application of: Plesko et al.
Application No. 10/607,591
Filed: June 27, 2003
Confirmation No. 5996
For: TYPE SYSTEM FOR REPRESENTING AND CHECKING CONSISTENCY OF HETEROGENEOUS PROGRAM COMPONENTS DURING THE PROCESS OF COMPILATION
Examiner: Thuy Chan Dao
Art Unit: 2192
Attorney Reference No. 3382-64706-01

INTERVIEW AGENDA – (Informal – Do not enter)

Attached is an Interview Agenda for the telephone interview that has been scheduled for Tuesday, November 18, 2008, concerning the above-referenced application, between Examiner Thuy Chan Dao and Mark W. Wilson, representing the applicants, at 1:30 p.m. EST (10:30 p.m. PST).

Applicants thank the Examiner in advance for the courtesy of the scheduled interview.

I. Claim 1

A. Claim 1 recites:

> A method of type-checking a code segment written in a programming language comprising:
>
> translating the code segment from the programming language to one or more representations of an intermediate language, wherein the one or more representations of the intermediate language are capable of representing programs

written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language; and

type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known.

B. Lidin, *Inside Microsoft .NET IL Assembler* ("IL Assembler")

IL Assembler's description at pages 9 through 12 does not describe "a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known" as stated in the Action at page 5.

C. Differences

IL Assembler does not teach or suggest "a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known" as recited by claim 1. Portions of *IL Assembler* pages 9–12 are reproduced below:

Native Types

When managed code calls unmanaged methods or exposes managed fields to unmanaged code, it is sometimes necessary to provide specific information about how the managed types should be marshaled to and from the unmanaged types. The unmanaged types recognizable by the common language runtime are referred to as *native*, and they are listed in CorHdr.h in the enumeration *CorNativeType*. All constants in this enumeration have names that begin with *NATIVE_TYPE_** ; for purposes of this discussion, I have omitted this part of the names or abbreviated it as *N_T_*. The same constants are also listed in the .NET Framework class library in the enumerator *System.Runtime.InteropServices.UnmanagedType*.

Some of the native types are obsolete and are ignored by the runtime interoperability subsystem. But since these native types are not retired altogether, ILAsm must have ways to denote them—and since ILAsm denotes these types, I cannot help but list obsolete types along with others, all of which you'll find in Table 7-6.

Table 7-6. Native Types Defined in the Runtime

Code	Constant Name	.NET Framework Type Name	ILAsm Notation	Comments
0x01	*VOID*		*void*	Obsolete and thus should not be used; recognized by ILAsm but ignored by the runtime interoperability subsystem
0x17	*FIXEDSYSSTRING*	*ByValTStr*	*fixed sysstring [<size>]*	Fixed-system string of size <size> bytes; applicable to field marshaling only
0x18	*OBJECTREF*		*objectref*	Obsolete
0x19	*IUNKNOWN*	*IUnknown*	*iunknown*	IUnknown interface pointer
0x1A	*IDISPATCH*	*IDispatch*	*idispatch*	IDispatch interface pointer
0x1B	*STRUCT*	*Struct*	*struct*	C-style structure, for marshaling the formatted managed types
0x1C	*INTF*	*Interface*	*interface*	Interface pointer

Variant Types

Variant types are defined in the enumeration *VARENUM* in the Wtypes.h file, which is distributed with Microsoft Visual Studio. Not all variant types are applicable as safe array types, according to Wtypes.h, but ILAsm provides notation for all of them nevertheless, as shown in Table 7-7. It might look strange, considering that variant types appear in ILAsm only in the context of safe array specification, but we should not forget that one of ILAsm's principal applications is the generation of test programs, which contain known, preprogrammed errors.

Table 7-7. Variant Types Defined in the Runtime

Code	Constant Name	Applicable to Safe Array?	ILAsm Notation
0x00	*VT_EMPTY*	No	*<empty>*
0x01	*VT_NULL*	No	*null*
0x02	*VT_I2*	Yes	*int16*
0x03	*VT_I4*	Yes	*int32*
0x04	*VT_R4*	Yes	*float32*
0x05	*VT_R8*	Yes	*float64*
0x06	*VT_CY*	Yes	*currency*
0x07	*VT_DATE*	Yes	*date*
0x08	*VT_BSTR*	Yes	*bstr*
0x09	*VT_DISPATCH*	Yes	*idispatch*
0x0A	*VT_ERROR*	Yes	*error*
0x0B	*VT_BOOL*	Yes	*bool*
0x0C	*VT_VARIANT*	Yes	*variant*
0x0D	*VT_UNKNOWN*	Yes	*iunknown*
0x0E	*VT_DECIMAL*	Yes	*decimal*

What *IL Assembler* describes are the known interface types iUnknown and iDispatch. These are *known* interface types that are defined for Microsoft COM object interfaces:

> "The **IUnknown** interface lets clients get pointers to other interfaces on a given object through the **QueryInterface** method, and manage the existence of the object through the **IUnknown::AddRef** and **IUnknown::Release** methods. All other COM interfaces are inherited, directly or indirectly, from **IUnknown**. Therefore, the three methods in **IUnknown** are the first entries in the VTable for every interface."

(Exhibit A, *MSDN COM fundamentals: IUnknown(COM)*, available at: http://msdn.microsoft.com/en-us/library/ms680509(VS.85).aspx). *IL Assembler* therefore teaches designating a type as *a known type* called "iUnknown," which designates the type as a well-known COM interface object type. Hence, *IL Assembler* does not teach or suggest designating a type as *an unknown type*, which "indicates that an element of the representation is of a type *that is not known*" as in the method of claim 1.

II. Claim 6.

A. Claim 6 recites:

> A method of selectively retaining type information during compilation in a code segment written in a programming language, the method comprising:
>
> translating the code segment from the programming language to one or more representations of an intermediate language;
>
> for each representation, determining whether to retain type information for one or more elements of the representation;
>
> based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type; and
>
> type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking the type designated as the unknown type.

B. U.S. Patent No. 6,560,774 to Gordon (Gordon)

Gordon does not teach or suggest determining whether to retain type information and associating elements with an unknown type. The Examiner contends that the language of claim 6, "determining whether to retain type information for one or more elements of the representation; based on the determination, associating one or more elements of the representation with a type, designated as an unknown type" is taught by *Gordon*. (Action, page 6).

C. Differences

Gordon column 27 and FIG. 23 describes a "Virtual Execution System" (VES) that can skip verification, including "native-size primitive type checks" of precompiled native code that is fully trusted. (*Gordon*, col. 3, lines 7–8; col. 6, line 67–col. 7, line 5; col. 27, lines 12–33). *Gordon* teaches that the VES does not maintain type information to aid verification: "the type of the arguments to any method in the code being verified *are fixed*. . . . This means that dynamic type information for arguments *do not need to be maintained* on a per-basic block basis." (*Gordon*, col. 7, lines 57–60) (emphasis added). *Gordon* further teaches that type information for primitive locals is "fixed" and that for non-primitives, type information must be maintained:

> [T]he type of primitive local variables, such as integers, floating points, etc., *is fixed*. If a variable is declared as being a primitive type, then it is not allowed to store a non-primitive type, such as an object reference, into it. In this manner, a single bit--"live" or "dead" for example--is sufficient to convey a primitive variable's state at any point. For non-primitive variables, however, *complete type information must be maintained*.

(*Gordon*, col. 7, line 66 through col. 8, line 6). This section does not describe "a type designated as an unknown type" nor does it disclose determining whether to retain type information for one or more elements of the representation, as recited by claim 6.

Gordon, therefore, discloses a method of verification that either (1) does not maintain dynamic type information for arguments or (2) maintains complete type information as designated in the source code. This is not the same as the method of claim 6, which, after determining whether to retain type information for elements of the intermediate language representation, designates one or more elements "as an unknown type."

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301
cc: Docketing

By _____
Mark W. Wilson
Registration No. 63,126

Exhibit A

http://msdn.microsoft.com/en-us/library/ms680509(VS.85).aspx

msdn

COM

IUnknown (COM)

The **IUnknown** interface lets clients get pointers to other interfaces on a given object through the **QueryInterface** method, and manage the existence of the object through the **IUnknown::AddRef** and **IUnknown::Release** methods. All other COM interfaces are inherited, directly or indirectly, from **IUnknown**. Therefore, the three methods in **IUnknown** are the first entries in the VTable for every interface.

When to Implement

You must implement **IUnknown** as part of every interface. If you are using C++ multiple inheritance to implement multiple interfaces, the various interfaces can share one implementation of **IUnknown**. If you are using nested classes to implement multiple interfaces, you must implement **IUnknown** once for each interface you implement.

When to Use

Use **IUnknown** methods to switch between interfaces on an object, add references, and release objects.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface [http://msdn.microsoft.com/en-us/library/ms682521(VS.85).aspx]	Returns pointers to supported interfaces.
AddRef [http://msdn.microsoft.com/en-us/library/ms691379(VS.85).aspx]	Increments reference count.
Release [http://msdn.microsoft.com/en-us/library/ms682317(VS.85).aspx]	Decrements reference count.

Requirements

For an explanation of the requirement values, see Requirements (COM) [http://msdn.microsoft.com/en-us/library/ms693432(VS.85).aspx] .

Windows NT/2000/XP: Requires Windows NT 3.1 or later.

Windows 95/98: Requires Windows 95 or later.

Header: Declared in unknwn.h.

Send comments about this topic to Microsoft.

Tags:

Community Content